Analytical Approach to Layout Generation of Datapath Cells

Maciej Ciesielski, Serkan Askar, Samuel Levitin †
{ciesiel, saskar,}@ecs.umass.edu, sam.levitin@intel.com
Department of Electrical & Computer Engineering
University of Massachusetts, Amherst, MA 01003
† Intel Corporation, Shrewsbury, MA 01545

Abstract—
This paper addresses the problem of layout automation of datapath cells. It presents an analytical approach to transistor placement under full custom design style and demonstrates that it can be applied to practical datapath designs. The presented approach is based on a mathematical model which employs a mixed integer linear programming (MILP) technique. The placement algorithm adopts the custom design techniques commonly used by datapath layout designers in generating hand-crafted designs. An important aspect of the presented method is the efficient management of the complexity of the underlying mathematical model, which makes it applicable to real designs. We implemented the presented datapath design technique as an experimental software tool running in the industrial environment. The generated layout results are competitive with manual designs provided by experienced layout designers.

I. Introduction

Datapaths are characterized by a highly regular layout structure. A typical datapath floorplan consists of an array of horizontally oriented words of identical bit cells, called *datapath cells* and vertically oriented *bit slices*. Since each bit slice is replicated a number of times (determined by the datapath width) with very little or no modification, layout generation of such regular structures reduces to a careful design, often by means of hand-crafting, of individual datapath cells.

This paper addresses the problem of layout design automation of datapath cells, and specifically concentrates on the component placement problem using a *diffusion-limited* model. In this model, the components (individual transistors, chains of transistors, logic gates, etc.) are spaced in such a way as to satisfy the diffusion-to-diffusion physical design rules, without regard to detailed routing requirements. We concentrate on the full custom design style which has been commonly used in hand-crafted designs of datapaths. Here the task of datapath layout optimization amounts to minimizing the area of the datapath cell, while taking into account some routability and performance constraints.

The flexibility and great degree of freedom in datapath device placement makes the datapath cell design difficult to automate [1]. As a result, datapaths did not enjoy the benefits of layout design automation as did the gate array or standard cell based designs of control and random logic. Considerable effort has been devoted to automate layout synthesis in full custom environment for general building-block placement [2], [3], but little documented work has been achieved in datapath synthesis. It has been our ambition to provide an automated, custom-quality placement tool for datapath cells, with a level of compaction comparable with those hand-crafted by experienced layout designers, but achieved in a significantly shorter amount of time and with predictable results. The method presented in this paper is based on known mathematical programming techniques, but its contribution lies in the efficient mathematical modeling and the management of the complexity of the placement problem.

This work has been supported by a grant from Compaq Computer Corporation, Alpha CAD & Test Group

II. Background and Previous Work

A. General Placement Techniques

The placement problem has been widely researched for all design styles [4], [1], [5], [6], [7], [8]. The placement algorithms can be broadly divided into two classes: *deterministic* and *stochastic*. The class of **deterministic algorithms** include numerical, constructive, and analytical methods, briefly described below.

- **Numerical methods** solve the placement problem as a system of simultaneous equations for all components at once. A typical example of such a method is *force-directed method*, which models the placement problem as a mechanical system of objects connected by springs [4], [9], [10]. Additional forces can be introduced to minimize the amount of component overlap [11]. Unfortunately, datapath placement cannot benefit from the efficiency of these techniques due to dissimilar geometry of modules in datapath cells and other requirements imposed by full custom design.
- **Constructive methods** generate a solution by placing one component at a time. A multitude of constructive methods have been developed for different design styles, including the full custom design. Most of them are based on solving a 2-D bin packing problem [12] [13]. In contrast to force-directed methods, these methods are mostly applicable to geometrical packing problems, where the connectivity constraints are not critical.

• **Analytical methods** are based on mathematical programming techniques, such as *Mixed Integer Linear Programming* (MILP) and *Quadratic Programming* (QP). Several methods have been proposed to solve the placement problems with both rigid and flexible modules using integer variables to represent the relative positions of placed components [14], [15], [7] [15]. The run time of this class of methods is exponential in the number of integer variables used in the model, which, in general, makes it impractical for designs with a large number of components. However, its optimization power makes it attractive for those problems where the complexity of the input space can be made manageable. We shall use this model in our work to derive component placement by finding efficient representation of component relations to control the complexity of the underlying combinatorial problem. Another group of analytical methods uses *resistive network* analogy to formulate the layout problems for standard cell and gate-array designs [16]. Since the placement problem can be viewed as a discrete mathematical optimization problem, it has been also approached using different forms of *branch-and-bound* technique [17]. Recently, the *sequence pair* approach was found to be very efficient in rectilinear module placement [18]. Quadratic Programming and spectral, eigenvalue-based methods have been also used to compute placement, but they are mostly applicable to semi-custom design styles [19].

The following **stochastic methods** are commonly used to model the layout design problems.

• **Simulated annealing** (SA) approach is analogous to hardware annealing process. It gained wide acceptance among researchers due to its ability to handle multi-dimensional cost function [20], [21], [22]. Place and route tools, such as KOAN/ANAGRAM II [23] and PUPPY [24], designed for analog devices, are best examples of layout techniques based on simulated annealing. A mixed transistor/gate level placement tool for digital datapaths, AKORD [25], also uses SA to optimize different objectives such as area and connectivity under the specific custom design constraints.

• **Genetic algorithms** (GA) have also been proposed for different placement and floorplanning problems [26], [27]. The genetic algorithm is a search technique that emulates the natural process of evolution as a means of progressing toward an optimum solution.

Despite the wealth of literature on the placement problem, there has been little documented work on custom-quality datapath synthesis reported in literature. While most CAD companies today use datapath layout generators in their tool suites, these tools are proprietary and tuned to their particular product environment. The problem of datapath cell layout design has been specifically addressed in [25] and [28]. A simulated annealing based placement tool, AKORD [25] generates compact layouts comparable to manual design for most of the tested datapath examples. However, for each design, fine-tuning of the cooling schedule and the associated SA parameters increase the turnaround time between the experiments. Moreover, the risk of generating layouts with minor design rule violations, a feature typical of SA-based algorithms for this class of problems, makes AKORD less attractive to datapath synthesis. A constructive method reported in [28] approaches the layout design of datapath cell as a 2-D bin packing problem and ignores to a large extent the interconnect issues. Furthermore, custom design techniques, such as transistor folding and merging, cannot be modeled efficiently with this approach.

Our approach to datapath transistor placement problem is based largely on Mixed Integer Linear Programming (MILP) technique, as it offers a good compromise between the result quality and model complexity. Because of its inherent exponential complexity, we carefully define the granularity of the model, which makes it possible to obtain quality layouts in a reasonable CPU time.

B. Datapath Placement

Datapath circuits are typically organized in (horizontal) rows of *words* representing the same functional block and (vertical) *bit slices*, delimited by vertically running power and ground rails. A typical organization of a datapath is shown in Figure 1. Each word is composed of a number of identical *datapath cells* placed next to each other, side by side, separated by power and ground rails. With the exception for the boundary cells, the layout of the datapath cell of bit slice i is identical to that of bit slice $(i+1)$, but mirrored along the vertical axis so that the adjacent bit slices can share common power or ground rail.

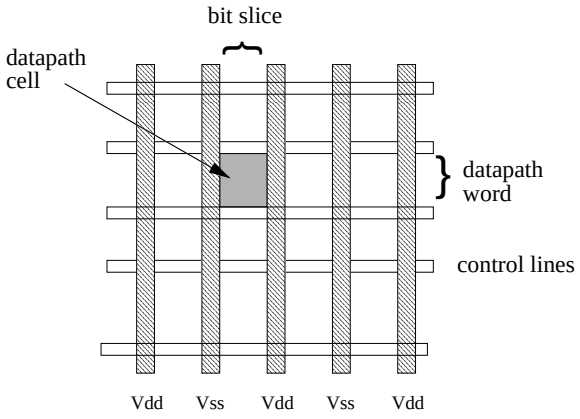


Fig. 1. Global floorplan of a datapath

Figure 2 shows an outline of a single datapath cell in a reference horizontal orientation. The width of the bit slice, also known as a *pitch*, is fixed; it determines the width for all the datapath cells. Power and ground (VDD/VSS) supply rails generally delimit the pitch. Signal nets are connected to the datapath cell components by means of *bristles*. Vertical bristles, or *data lines*, provide wiring between different cells within the same bit slice. They run in parallel to the power rails. Horizontal bristles, or *control lines*, provide wiring between datapath cells of different bit slices. Control lines span the width of the datapath and run perpendicular to the power rails. The physical location of the bristles is typically fixed prior to placing

the transistors in the datapath cell. In addition to purely

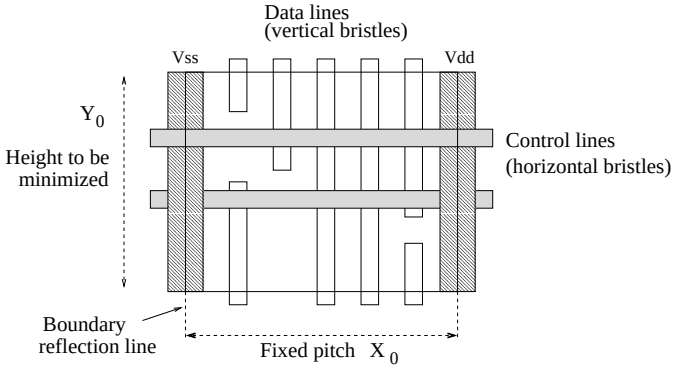


Fig. 2. Representation of a datapath cell

geometrical constraints imposed by device sizes annotated in the circuit diagram, our approach considers several practical constraints adopted from full custom, hand-crafted layout design practices. These include *transistor chaining* and *folding*, and *device merging*.

• **Transistor chaining and device merging.** Transistor chaining is a widely used technique to improve both area and performance of datapath cells. Several transistors can be chained together by combining their diffusion areas in order to reduce the diffusion capacitance [29]. When the chained devices have different widths, the resulting component can take the form of a rectilinear polygon as shown in Fig. 4. Diffusion sharing applied to simple logic gates in the same datapath cell is known as *device merging*. In our system device merging reduces the total number of components to be placed, and is generated as part of our preprocessing scheme. In an attempt to optimize performance, dynamic nodes are considered as prime candidates for merging.

• **Transistor folding.** Transistor folding is another popular technique aimed at minimizing area and improving performance of custom designs [30], [31]. The folding changes the aspect ratio of the component, while maintaining the required device size (W/L ratio). By performing folding with different number of *fingers* (poly gates), different component instances can be created for the placement phase. The folding of devices is done as part of the preprocessing phase, described in Section III-A.

• **Intra-cell sharing.** Two component areas (diffusion regions or poly gates) belonging to components from adjacent bit slices can be merged if they share the same *global* net, such as power or ground line, control line, or clock signal. In a typical organization of a datapath, adjacent bit slices are identical copies of each other, reflected with respect to the vertical boundary line. In this case the components can be pushed under the boundary line (power or ground rail) and their diffusion regions (or poly gates) merged, creating a more compact layout, as shown in Figure 3. This type of diffusion sharing further contributes to a reduction in load capacitances, a very desirable feature in high-performance datapath designs.

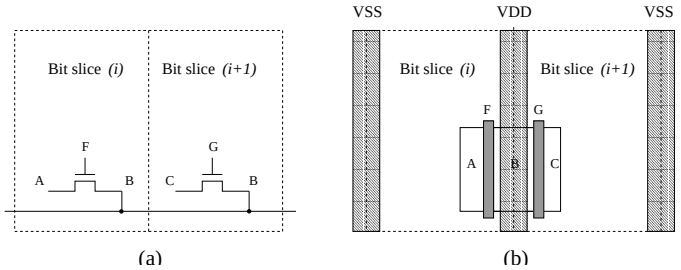


Fig. 3. Diffusion sharing between two components from adjacent bit slices

III. Our Approach to Datapath Placement Problem

The primary objective of datapath cell design is to minimize the layout area. For a fixed pitch, X_0, this problem reduces to minimizing the height of the datapath cell, Y_0 (refer to Figure 2). The performance and routability of the cell are considered as secondary objectives. It is reasonable to assume that the size of the cell is small enough to ignore the issue of interconnect delay introduced by wiring. In our system the performance issues are indirectly taken into consideration by performing on-the-fly device merging and sharing, as described earlier. These and other routability issues can be also addressed by minimizing some measure of interconnect complexity of the internal signal nets, incorporated in the form of additional, explicit constraints. In case of a bulk process (single-well or twin-well technology), the number of diffusion wells must also be minimized, as this improves yield and reduces the number of well plugs required.

In our approach each component in the datapath cell represents a single device (transistor or simple logic gate), a chain of transistors, or a group of devices merged together and to be placed as one entity. We refer to such single or combined devices as *placeable components*. Physical constraints imposed on placeable components allow for a certain degree of freedom, including multiple instances and orientations.

• **Multiple instances.** Each placeable component can take one of several allowed shapes, or *instances*. In our case the shapes are limited to rectilinear polygons. The different instances of placeable components are obtained either by transistor chaining (supplied by the circuit designer), or by transistor folding (generated as part of our procedure). The placement optimization procedure will automatically select the best instance of the component.

• **T-shaped and L-shaped components.** Rectilinear components can be modeled as sets of abutting rectangles, as shown in Figure 4.

• **Component orientation.** Each component is allowed to take an arbitrary orientation along the horizontal and vertical axes. Rectangular components can have two possible orientations, while the L-shaped and T-shaped components can assume any of the eight possible orientations. The system will automatically select the orientation that minimizes the overall cost function as part of the optimization.

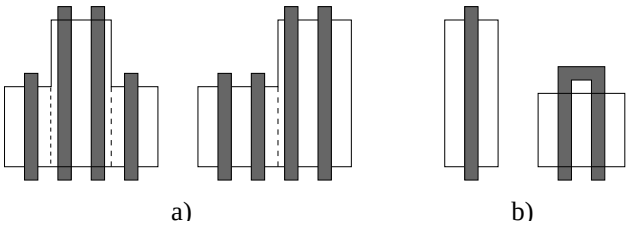


Fig. 4. Instances of placeable components: a) predefined shapes (chained transistors): T-shape and L-shape; b) generated instances (folded transistors)

Finally we must consider the minimum spacing between the component boundaries that satisfies the design rules of the given process technology.

In order to manage the complexity of datapath transistor placement problem the entire design process is divided into the following phases.

1. **Pre-processing.** In this step the connectivity, component geometry, bit slice geometry, and process technology information is obtained from the raw input files and stored in the database. Based on the component geometry and connectivity information, the possible device merging and folding is carefully examined and the multiple instances of placeable components are generated.
2. **Initial Relative Placement.** The goal of this phase is to derive a relative initial placement of components that will facilitate the modeling of the subsequent geometric placement problem. It uses a force-directed placement technique to find the relative positions of components based on their connectivity, while ignoring, for now, the component geometry.
3. **Component Grouping.** An important step in our procedure is the *grouping* of components into larger clusters to facilitate the subsequent geometric placement. This step is dictated by a need to limit the number of integer variables in the analytical formulation of the geometric placement.
4. **Geometric Placement.** This phase concentrates on the generation of non-overlapping placement of components, taking into consideration their geometries and physical design rules. The component connectivity is addressed implicitly by maintaining the relative positions of the components determined during the initial placement phase. The objective of the geometric placement is to minimize the height of the layout.
5. **Post-processing.** This phase addresses the routability and manufacturing issues, and specifically the minimization of wire length and diffusion wells. The geometrical placement computed in the previous phase is modified here by applying a series of size-invariant transformations, such as mirroring and swapping of components. The metrics for these transformations include total (or critical) wiring length minimization and, for a single-well technology, the minimization of the number of wells.

A. Pre-Processing

The goal of the pre-processing step is to process and analyze: the connectivity between the components, bit slice geometry, component geometry, and process technology. This information is then used to generate multiple instances of placeable components by means of transistor folding and device merging.

Connectivity information of the datapath cell is given as a wirelist, extracted from the circuit structure. Particular attention is being paid to *dynamic* nets, connected to the dynamic nodes or transmission gates. They are examined carefully in an attempt to shorten their wire length and to reduce the load capacitance by means of merging. Component geometry is provided by specifying the dimensions of the modules. Bit slice data includes the width of the slice (a *pitch*) and the location of the power rails. Finally, the technology file provides the design rules for the given process technology.

Merging and Folding. Our heuristics for merging and folding are based on techniques used by experienced layout designers in generating hand-crafted datapath layouts. The merging is employed as a means to provide multiple instances for the placeable components. If no merging is possible, the folding is applied instead. If the transistor contains a dynamic net, only the instances with even number of fingers are allowed as this minimizes the diffusion capacitance of the dynamic node more effectively.

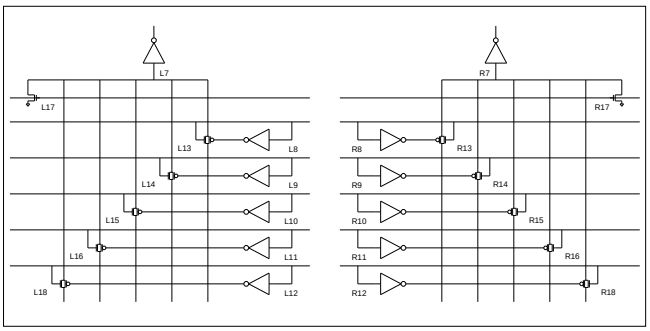


Fig. 5. Schematic of example 2X spanning two bit slices

Example 1: *2X is a datapath circuit with 24 components which spans two bit slices. The schematic of the cell is shown in Figure 5.*

The labels L and R indicate the left and right bit slice components. Most of the components are connected to the dynamic nets at the inputs to L7 and R7. Two pairs of transmission gates are merged at the dynamic nodes: L13 with L14, and L15 with L16. The N diffusion of the transmission gate L18 is merged with an NMOS device L17 at the dynamic node. Two pairs of inverters, (L8, L10) and (L11, L12), share common diffusion on power nodes and hence they are also merged. The same merging applies to the right bit slices. There is also merging between the components of the L and R parts. The large inverter L7 merges with R7 and the smaller inverter L9 with R9. As a result of the merging, the number of placeable components

was reduced from 24 to 12; the load capacitances on the dynamic nodes were also reduced. □

B. Initial Relative Placement and Component Grouping

The goal of this phase is to derive a *relative* initial placement of components that will facilitate modeling of the subsequent geometric placement problem. Connectivity is the major factor considered in this step, while component geometries are temporarily ignored; they will be taken into account in the geometric placement step. Standard force-directed technique [8] is used to compute the locations of the component centers, based on the connectivity of components to other components, bristles, and power rails.

In order to avoid a trivial solution (with all components placed on top of each other) the location of some of the components must be temporarily fixed [9]. We select a minimum number of such components, called *anchors*, one per each corner of the physical design space. The x coordinates for the anchors are known from the fixed pitch of the datapath cell, while their y coordinates are computed from the estimated cell height. Exact locations are not important in this phase; the location of the anchors will be relaxed during the subsequent geometric placement to meet the geometric constraints and minimize the cell height. Anchors are selected heuristically by analyzing the following factors:

1. *Component area and geometry:* mimicking the manual layout, large components placed in the corners of the datapath cell are considered as good candidates for anchors.
2. *Connections to bristles and power rails:* the connection to bristles and power rails, whose exact locations within the datapath cell are known, are taken into account in the initial placement algorithm. This helps address the routability issue and reduce the net length in the final layout.
3. *Shareability across the cell boundaries:* If a component shares diffusion with another component in the adjacent bit slice, it will be chosen as an anchor. It will then be placed close to (or under) the cell boundary during the geometric placement phase.

Example 2: *Figure 6 shows the result of the initial relative placement for the Example 2X. Component 35 is created by merging two inverters (L7, R7) of both bit slices. It has the largest area and is chosen as one of the bottom anchors. Anchors 27 (R15, R16) and 30 (L13, L14) are the next largest in area, with connections to north and south bristles, respectively. The fourth anchor is component 26 (R12, R11).* □

The computed initial placement provides an important information about the *relative* positions of the component centers. This information will be used in modeling of the subsequent geometric placement phase, with an integer variable Q_{ij} introduced to represent a selection of a relative position for a pair of components (i, j) (refer to Section III-C.3 and Figure 9). To further simplify the complexity of the geometric placement phase, the components are

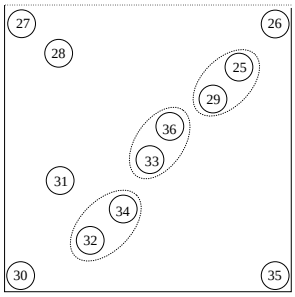


Fig. 6. Initial relative placement, with possible component grouping, for example 2X

grouped according to their proximity in the initial relative placement phase. This effectively reduces the number of integer variables Q_{ij} in the model (which is quadratic in the number of components). By combining components into groups, only one integer variable is needed to represent the relative position of the components in the group with respect to any other group, hence significantly limiting the total number of integer variables needed. The components inside each group are still placed independently, but maintain their relation to other components as a group, rather than individually.

The component grouping employs a simple clustering technique based on the *neighborhood* concept. Two components are neighbors if they are adjacent in the initial placement in terms of their x or y coordinates. If two components are neighbors in both directions, the pair is chosen to form a *group*. Each group is then considered as a new component, whose coordinates are derived from that of its members. The search for new groups continuous iteratively until the maximum allowed number of integer variables is reached. The grouping is not performed if the estimated number of integer variables is within the allowed limit (see the discussion in Section IV).

Example 3: *Dotted ovals in Figure 6 illustrate the possible grouping of components for the 2X example. In this case the grouping is actually not needed since the number of integer variables is sufficiently small.* □

C. MILP Formulation of Geometric Placement

C.1 Component Modeling

Each rectangular component i is modeled as a rectangle whose coordinates of the top/right corner are denoted by a pair (X_i, Y_i) as shown in Figure 7. The width W_i and height H_i of component i are defined as its dimensions perpendicular and parallel to the poly line, respectively. An integer variable R_i is introduced for each component i to model its *geometric orientation.* R_i=0 if the component is placed horizontally, and R_i=1, if it is placed vertically (see Figure 7).

Each rectangular component can assume up to two different shapes (instances). For this purpose, a new integer variable S_i, the *instance selection parameter*, is introduced for component i, such that S_i=0 if the first instance of the component is chosen, and S_i=1 if the

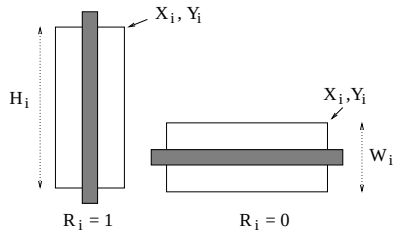


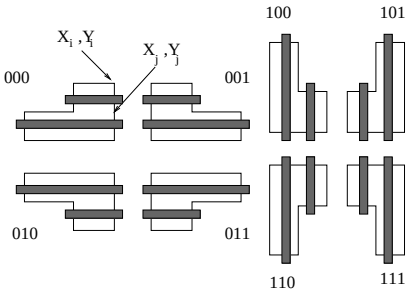
Fig. 7. Model of a component in two different orientations

second instance is chosen. This is modeled by the following equations [7]:

$$W_i = W_{i0} \cdot (1 - S_i) + W_{i1} \cdot S_i \quad (1)$$
$$H_i = H_{i0} \cdot (1 - S_i) + H_{i1} \cdot S_i \quad (2)$$

where W_{i0} and H_{i0} are the width and the height of the first instance and W_{i1} and H_{i1} are the width and the height of the second instance of component i. The model can be readily extended to multiple instances (shapes) at a cost of adding new integer variables. Larger number of component shapes would provide a greater flexibility in finding a compact datapath layout. However to keep the complexity low, we only allow for 2 instances of each placeable component.

In addition to rectangular components, our model also allows for L-shaped or T-shaped components. An L-shaped or T-shaped component is modeled as a set of abutting rectangles with the same poly orientation. During the geometric placement all eight orientations of these components are considered and modeled as a set of linear equations. The following constraints model all the possible



R	Mx	My	$X_i - X_j$	$Y_i - Y_j$
0	0	0	0	W_i
0	0	1	$H_i - H_j$	W_i
0	1	0	0	$-W_j$
0	1	1	$H_i - H_j$	$-W_j$
1	0	0	W_i	$H_i - H_j$
1	0	1	$-W_j$	$H_i - H_j$
1	1	0	W_i	0
1	1	1	$-W_j$	0

Fig. 8. The modeling of L-shaped components

orientations of an L-shaped component shown in Figure 8.

$$\begin{aligned} X_i - X_j = \\ (M_{y_j} - M_{y_j}R_j - M_{x_j}M_{y_j} + M_{x_j}M_{y_j}R_j) \cdot (H_i - H_j) \\ +(R_j - M_{x_j}R_j - M_{y_j}R_j + M_{x_j}M_{y_j}R_j) \cdot W_i \\ +(M_{x_j}My_j - M_{x_j}M_{y_j}R_j) \cdot (H_i - H_j) \\ +(M_{y_j}R_j - M_{x_j}M_{y_j}R_j) \cdot (-W_j) \\ +(M_{x_j}R_j - M_{x_j}M_{y_j}R_j) \cdot W_i + M_{x_j}M_{y_j}R_j \cdot (-W_j) \end{aligned} \quad (3)$$

where M_x and M_y are the integer variables modeling the mirroring along the x and y axis, respectively, and R_i is the orientation parameter for rectangle i. Similar equations are used for T-shaped components with only minor modifications. The product of binary variables M_{x_j}, R_j can be linearized as follows:

$$\begin{aligned} M_{x_j}R_j &\leq M_{x_j} \\ M_{x_j}R_j &\leq R_j \\ M_{x_j}R_j &\geq M_{x_j} + R_j - 1 \end{aligned} \quad (4)$$

Similar equations hold for the product $M_{x_j}M_{y_j}R_j$ and for the Y coordinate constraints.

C.2 Boundary Constraints

Each component must be placed within the boundaries of the datapath cell, determined by its fixed pitch, X_0, and variable height, Y_0 (to be minimized). Let $\triangle_{bi}$ be a distance from the boundary of component i to the respective boundary of the datapath cell. The boundary constraints are given as follows:

$$X_{Dim_i} + \triangle_{bi_x} \leq X_i \leq X_0 - \triangle_{bi_x} \quad (5)$$
$$Y_{Dim_i} + \triangle_{bi_y} \leq Y_i \leq Y_0 - \triangle_{bi_y} \quad (6)$$

where $\triangle_{bi_x}$ and $\triangle_{bi_y}$ are the margins to the vertical and horizontal boundary lines, respectively, and X_{Dim_i} and Y_{Dim_i} are given as

$$X_{Dim_i} = \begin{cases} W_i & \text{if } R_i = 1 \\ H_i & \text{if } R_i = 0 \end{cases} \quad Y_{Dim_i} = \begin{cases} H_i & \text{if } R_i = 1 \\ W_i & \text{if } R_i = 0 \end{cases} \quad (7)$$

Here W_i and H_i are the width and height of component i. If component i is shareable across vertical reflection line, $\triangle_{bi_x}$ can be negative to allow for intentional diffusion overlap. Otherwise $\triangle_{bi_x}$ is positive, its value determined by the respective physical design rule. Sharing across the horizontal reflection line is not permitted in our case, hence $\triangle_{bi_y} \geq 0$.

C.3 Non-overlapping Constraints

To generate a non-overlapping placement the distance between each pair of components must satisfy the diffusion spacing requirements, specified by the physical design rules. The initial relative placement limits the relationship between a pair of components to one of the non-convex areas of the design space: top *or* right, top *or* left, bottom *or* right, or bottom *or* left. For example, in the initial placement shown in Figure 6 component 25 is placed *above* and to the *right* of component 29. This allows us to fix the relationship between them so that 25 will be placed above *or* to the right of 29. To model this non-convex design space, a binary integer variable Q_{ij} is introduced for each pair of components (ij). The non-overlapping constraints take the following form:

$$X_j - X_i \geq X_{Dim_j} - L * (1 - Q_{ij}) + \triangle_{sep} \quad (8)$$
$$Y_j - Y_i \geq Y_{Dim_j} - L * Q_{ij} + \triangle_{sep} \quad (9)$$

Here L is a sufficiently large positive number, and $\triangle_{sep}$ is the required spacing between the components. With this formulation, the value of variable Q_{ij} fixes the physical relation between components i and j to be either horizontal

or vertical. This allows a component to be placed in three of the four quadrants with respect to its neighbor. As shown in Figure 9, for Q_{ij}=0 component j will be placed above component i, with no constraints imposed on its horizontal position with respect to component i. For Q_{ij}=1, component j will be placed to the right of component i, with no constraint on its vertical position. As a result, component j can be located anywhere above or to the right of i (but not below and to the left, as this would violate the initial relative placement condition).

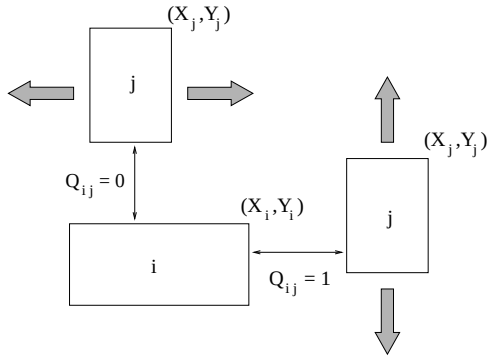


Fig. 9. Modeling of non-overlapping constrains

C.4 The Optimization Problem

The final optimization problem is to *minimize* Y_0 subject to the set of linear constraints (5 - 9) defined above. We solve this problem using a commercial MILP solver, CPLEX [32].

A typical datapath cell placement problem involves 20-50 original components. After component merging, the size of the typical problem reduces to less than 20 placeable components, and is further divided into groups. To make our approach computationally feasible, a limit has been imposed on the number of integer variables in our MILP formulation. We observed that MILP can run efficiently (and generate good result within several minutes on a standard PC) if it is limited to about 60 integer variables. This number, although problem-specific and varying from one optimization instance to another, has been quite stable for the class of problems addressed here. It has been determined empirically based on a large number of runs.

We control the complexity of the underlying optimization problem in several ways, including the merging and grouping of components, discussed earlier. Both of these techniques effectively reduce the total number of integer variables to the desired level. We should emphasize that the placement is carried out for *all* components at once, while restricting relative positions of components based on their group membership.

Example 4: *The result of geometric placement for example 2X is given in Figure 10. The impact of the initial relative placement on the physical locations of components is noticeable. See, for example, how component 25 was placed above 29 and pushed to the left of it.* □

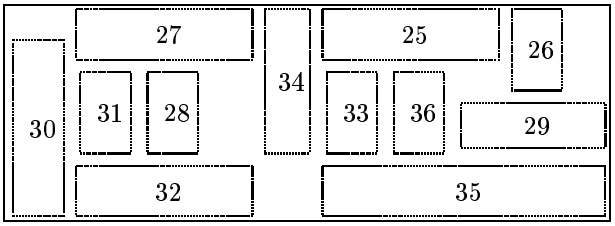


Fig. 10. Geometric placement of example 2X

D. Post-Processing

The goal of the post-processing phase is to generate the final layout of the placed transistors at the mask level. This phase deals with several fine-grain layout issues, such as determining the final positioning and orientation of the placed components, assignment of N wells to P diffusion regions, etc. This is accomplished by mirroring the devices along the X and Y axes. Notice that such mirroring affects the absolute positions of component pins and the location of the P/N diffusion regions, and therefore can be used to reduce the number of wells and to simplify the routing. Minimizing the number of wells is an important issue from the manufacturing point of view for the bulk process. Once the design meets the required timing specification, it is more important to minimize the number of wells than to reduce the net length, according to the experienced layout designers. The well minimization also contributes to routability by minimizing the number of well ties.

Our approach to final layout generation is composed of two steps. First, we compute the orientations of the components to minimize the number of wells. This information is then used as a constraint in the next step which globally minimizes the weighted net length.

D.1 Well Minimization

We assume here a single well technology and formulate the N-well minimization problem as a global optimization problem and solve it using a MILP approach.

The first step in our procedure is to represent all possible P diffusion region adjacencies between the adjacent components. We will use an *interval representation* of the components, defined as follows: each component is represented by a horizontal and vertical *interval* by projecting the component onto the X and Y axis, respectively. The length of the horizontal interval gives the width, while the length of the vertical interval specifies the height of the component. Intervals that are adjacent in one direction and overlap in the other represent adjacent components. For example, the vertical intervals 1 and 3 overlap while horizontal intervals 1 and 3 are adjacent, indicating that components 1 and 3 are adjacent in the layout. The overlap between intervals in either direction determines a possible adjacency between their N or P diffusion regions. By analyzing the adjacency information in the interval representation, we can identify the pairs of components that may contribute to the minimization of wells.

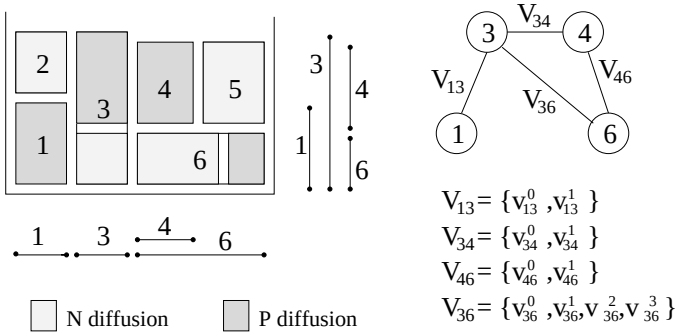


Fig. 11. Obtaining the diffusion adjacency graph from geometric placement

The diffusion adjacency is modeled by a *Diffusion Adjacency Graph (DIAG)*. Each node in the graph represents a component; an edge (i,j) is created for each pair of components that have adjacent *overlapping intervals* in either direction. Figure 11 shows an example of a DIAG for a sample layout. Each edge $(i,j) \in DIAG$ is assigned a set of *weights* representing the amount of the diffusion overlap, $V_{ij} = \{v_{ij}^k\}$. The construction of weights v_{ij}^k is illustrated in Figure 12; the mirroring of component i along the X axis gives rise to two different values of P diffusion adjacency with component j, namely v_{ij}^0 and v_{ij}^1. For the graph in Figure 11, the set V_{36} contains four weights, v_{36}^0 through v_{36}^3, due to four different combinations of diffusion regions of components 3 and 6.

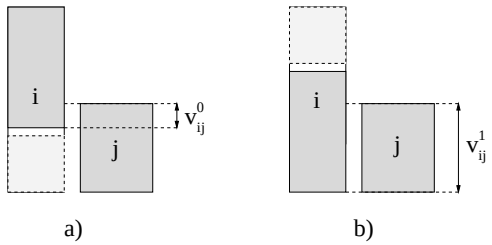


Fig. 12. Computation of interval overlaps: a) original position; b) component i mirrored along X axis

To model the problem in analytical terms we introduce a binary variable M_i to represent the mirroring of a double-diffusion component i along its respective axis. For single-diffusion components there is no need to introduce such variables since the mirroring will have no effect on the diffusion placement. Variable M_i will refer to the mirroring of component i along the X axis if the orientation of the component (as computed in the geometric placement) is vertical (it has rotation parameter $R_i = 1$); otherwise, for $R_i = 0$, M_i will refer to the mirroring along the Y axis.

Assuming that only one component (i) has both diffusion types, the interval overlap Z_{ij} for a component pair (i,j) can be written as follows:

$$Z_{ij} = (1 - M_i) \cdot v_{ij}^0 + M_i \cdot v_{ij}^1 \quad (10)$$

where v_{ij}^0 and v_{ij}^1 are the overlap weights for $M_i = 0$ and 1, respectively (refer to Figure 12). The above equation can easily be extended to the case when both components have double diffusions:

$$Z_{ij} = (1 - M_i - M_j + M_{ij}) \cdot v_{ij}^0 + (M_i - M_{ij}) \cdot v_{ij}^1 + (M_j - M_{ij}) \cdot v_{ij}^2 + M_{ij} \cdot v_{ij}^3 \quad (11)$$

The product $M_{ij} = M_i \cdot M_j$ can be linearized as in equation 4.

The objective is to maximize the sum of the interval overlaps between all pairs of components in the DIAG,

$$max \sum_{(i,j) \in DIAG} Z_{ij} \quad (12)$$

This maximizes the adjacency between the P diffusion regions, leading to a solution with minimum number of compact wells, as illustrated in Figure 13.

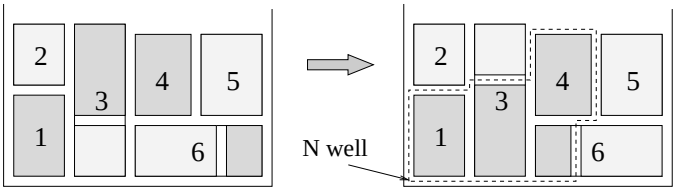


Fig. 13. Example of well minimization by component mirroring

Example 5: *Figure 14 shows the result of the well minimization for the circuit 2X, compared to the geometric placement in the reference orientation. The mirroring of the selected components improved the P diffusion overlaps in the final layout, resulting in fewer N wells.* □

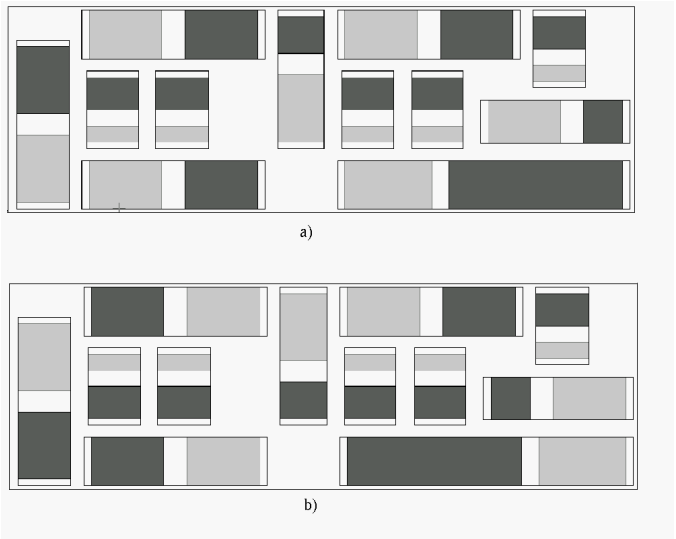


Fig. 14. Effect of well minimization for example 2X; a) before well minimization; b) after well minimization

D.2 Net Length Minimization

The length of a net is determined by the exact locations of its terminals (pins). Mirroring of components along the X or Y axis affects the location of the pins and hence the

length of the net connected to the pins. The goal of net length minimization is to find the mirroring of components that reduces the overall (weighted) net length. This is illustrated symbolically in Figure 15. We formulate the net length minimization problem as a global optimization problem over the entire set of wires. Each pin i is modeled

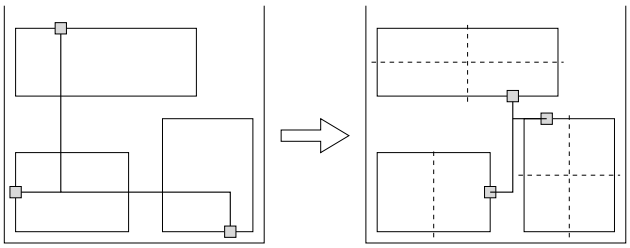
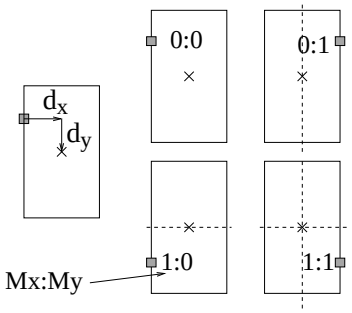


Fig. 15. Effect of mirroring on net length

as a point with coordinates (Xp_i, Yp_i). The location of the pin is a function of the component mirroring, as shown in Figure 16. We introduce two binary variables M_{x_i} and M_{y_i}



Mx	My	X_p	Y_p
0	0	$-d_x$	d_y
0	1	d_x	d_y
1	0	$-d_x$	$-d_y$
1	1	d_x	$-d_y$

Fig. 16. Pin location as a function of component mirroring in X and Y directions

per component i, to model the mirroring along the X and Y axis, respectively. The location of each pin is computed relative to the component center, which is not affected by mirroring. The location Xp_i of pin i on component q is expressed by the following linear equations:

$$Xp_i = Xc_q - (1 - M_{x_q} - M_{y_q} + M_{xy_q}) \cdot d_x + (M_{x_q} - M_{xy_q}) \cdot d_x - (M_{y_q} - M_{xy_q}) \cdot d_x + M_{xy_q} \cdot d_x \quad (13)$$

where Xc_q is the x location of the center of component q; d_x is the displacement of the pin from the component center in X direction; and $M_{xy_q} = M_{x_q} M_{y_q}$. The latter can be represented as a set of linear equations, similar to equation 4. The Y coordinates of the pins are derived in a similar fashion.

We approximate the length of a net using a *half perimeter* of the bounding box that includes all the pins of the net. The objective is to minimize the weighted sum of the lengths of all the nets, with higher weights assigned to *critical* nets. In order to preserve the solution of well minimization additional constraints are imposed by fixing the mirroring variables, M_{x_q} and M_{y_q}, for all the components q affected by well minimization.

IV. Results

The analytical technique for transistor layout generation described in this paper has been implemented as an experimental software tool. We tested our program on several datapath circuits made available by Alpha Development Group of Compaq. The complexity of those circuits ranged from 10 to 40 transistors, with the number of placeable components between 9 and 20. This reflects a typical complexity of datapath cells we have seen in industry. We compared our results in terms of datapath cell height with those obtained manually by an experienced layout designer. We are not aware of any other datapath layout synthesis tool for the purpose of fair comparison. We should recall that our datapath synthesis tool does not take into account the detailed routing. Nevertheless, for all the tested circuits the geometric placement generated by the tool was routable after minor modifications involving placing additional contacts required for routing. In some cases this required the expansion of the cell height to permit insertion of required contacts and wires, while preserving the cell topology.

		Height [λ]		CPU [min]
Circuit name	# of trans.	Custom design	Our approach	Our approach
PGK9	20	66	78.5	1:40
PB7	8	42	42	0:20
CS15	45	155.5	150	2:30
MU9	12	77.5	90	4:20
CMU8	9	180	190.5	0:20
2X	24	76	76	0:25

TABLE I
Comparison with hand crafted results

Table I compares the final (routed) layouts with the results obtained manually. Both the automatically generated layout and the custom layout were routed (manually) by the same designer. The datapath cell heights are given in design units, λ. The experiments were conducted on a 200 MHz Intel® Pentium ® processor running Linux. [1] A commercial MILP solver CPLEX [32] was used for solving the MILP-based optimization problems. The limit on the total number of variables in the MILP model was set to 60. This number has been determined empirically based on a large number of runs in order to finish the computation within a few minutes of CPU time. As can be seen from the table, our results compare favorably with the manual layouts, while taking only several minutes to compute.

CS15 is one example where our approach generated a better result than the designer. As the number of components in a circuit increases, the manual layout generation becomes less manageable; this in turn decreases the quality of the constructed datapath layout.

[1] Intel and Pentium are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

				Interval Overlaps [λ]		Net length [λ]	
Circuit name	# of trans.	# of nets	# of pins	Before optim.	After optim.	Before optim.	After optim.
PGK9	20	21	94	580	580	74598	71598
PB7	8	12	35	740	880	8746	8466
CS15	45	35	85	1080	1360	46640	43800
MU9	12	17	63	480	760	30780	28940
CMU8	9	14	96	620	760	68720	67140
2X	24	32	122	1120	1980	75444	74010

TABLE II
RESULTS OF POST-PROCESSING

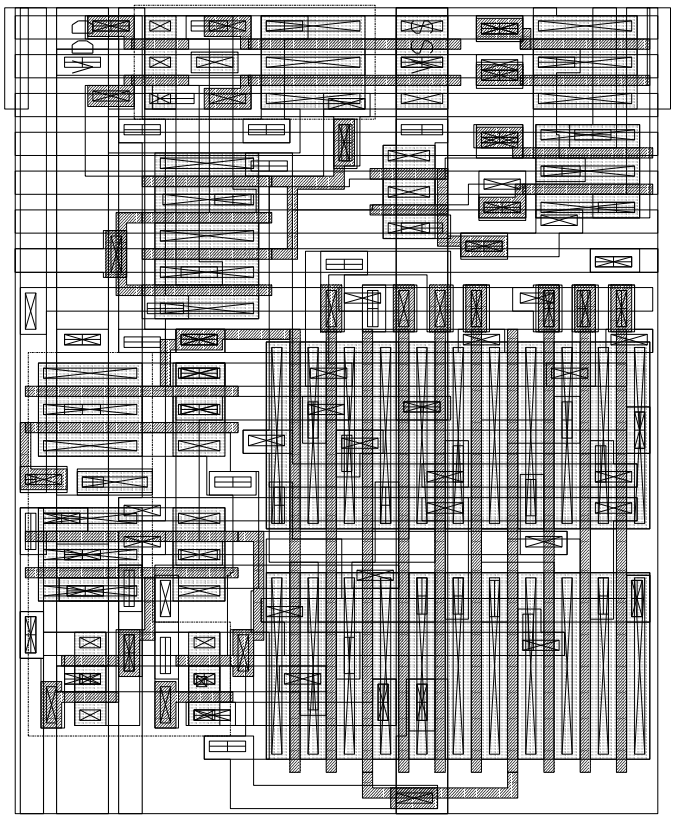


Fig. 17. Manually generated layout of example CS15 (routed)

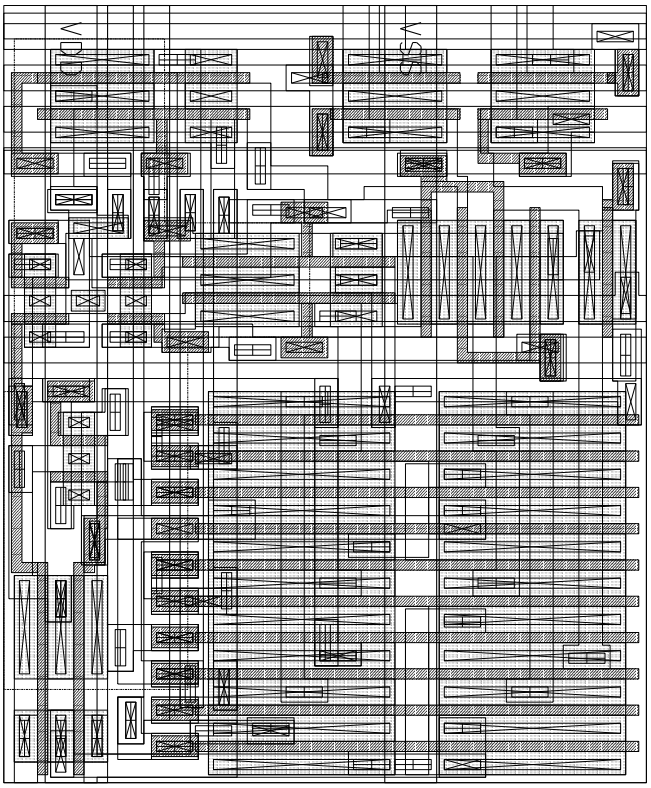


Fig. 18. Automatically generated layout of example CS15 (routed)

It is important to emphasize that the manually generated results include additional techniques and "tricks of the trade" such as unequal transistor folding, device splitting, etc. Designers often fold devices in the best possible way to fit them in the available layout slots. These techniques are currently not considered by our approach, which only considers rectangular shapes.

Table II compares the post-processing results for net length and well minimization with the results of geometrical placement. The table shows our measure of improved routability in terms of the overall net length. The amount of the interval overlaps (used as a measure of well packing) and the overall net lengths are given in λ. We believe that our post-processing step contributed to the routability of the tested datapath cells by minimizing the critical net length and the number of the required well ties.

V. Conclusions and Future Work

This work presents an attempt at automating datapath layout design. We presented a methodology and a practical software tool to generate datapath cell layout on a transistor level.

While most of the mathematical techniques employed here are not new, the contribution of this work lies in the efficient management of the complexity of the placement problem, which is inherently exponential. This comes in two places: 1) by dividing the problem into separate, yet related steps of a) initial relative placement (which

considers component connectivity only), and b) geometric placement (which respects the derived connectivity in terms of relative placement and computes non-overlapping placement); and 2) by limiting the number of integer variables Q_{ij} in the MILP formulation by grouping the components according to their connectivity (initial relative placement). As a result, the size of the problem in the number of integer variables remains manageable, and the placement problem can be solved in a matter of minutes on todays computers.

Although the routing was not taken into account explicitly, all the generated layouts were routable – either directly or with only minor modifications performed during manual routing. The results demonstrate significant time reduction within acceptable limits of area overhead compared to manual designs. The results are better for larger circuits, where human limitations at handling large design complexity becomes more apparent. The global post-processing optimization further improves the quality of generated layouts in terms of their routability and manufacturability. We are confident that our analytical approach combined with proper post-processing and iterative improvement can create automatically routable and fully acceptable datapath layouts competitive with those obtained by human designers, but at a significantly shorter time.

Future work in the datapath synthesis research might focus on the following issues:

- Develop a systematic way to include routability measures in the model. This can be achieved by adding a global routing phase to more accurately predict the effects of routing on circuit area.
- Develop better anchoring heuristics and make the method less sensitive to anchoring and less dependent on the initial relative placement.
- Consider improvements to MILP model by means of hierarchical placement and iterative group relaxation in order to handle larger designs.
- Improve post-processing by adding layout compaction involving miscellaneous geometrical transformations and component swapping.

Acknowledgments

The authors would like to thank Ken Slater, formerly of Compaq Computer Corporation, for invaluable comments on the methodology for datapath layout generation and his guidance regarding the tool development.

References

[1] B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, 1988.

[2] R. Rivest, "The 'PI' (placement and interconnect) system," *Proc. 19th Design Automation Conference*, pp. 475–481, 1982.

[3] N. P. Chen, C. P. Hsu, E. S. Kuh, C. C. Chen, and M. Takahashi, "BBL: A building block layout style for custom chip design," *Proc. IEEE Int. Conf. on Computer Aided Design*, pp. 40–41, 1983.

[4] M. Hanan and J.M. Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems*, M. A. Breuer, Ed., pp. 213–282. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1972.

[5] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley-Teubner Series, 1990.

[6] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993.

[7] M. Sarrafzadeh and C. K. Wong, *An Introduction to VLSI Physical Design*, McGraw-Hill Series, 1994.

[8] M. S. Sait and H. Youssef, *VLSI Physical Design Automation*, IEEE Press - McGraw-Hill, 1995.

[9] N. R. Quinn and M. A. Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Transactions on Circuits and Systems*, pp. 377–388, 1979.

[10] H. Onodera, M. Sakamato, T. Kurihara, and K. Tamaru, "Step by step placement strategies for building block layout," *Proc. Int. Symp. on Circuits and Systems*, pp. 921–926, 1989.

[11] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," *35th ACM/IEEE Design Automation Conference*, pp. 269–274, 1998.

[12] B. Baker, "Orthogonal packing in two dimensions," *SIAM Journal on Computing*, pp. 846–855, 1980.

[13] B. Chazelle, "The bottom-left bin packing heuristic: An efficient implementation," *IEEE Transactions on Computers*, 1983.

[14] L. Markov, J. R. Fox, and J. H. Blank, "Optmization techniques for two-dimensional placement," *Proceedings of 21th Design Automation Conference*, pp. 655–656, 1984.

[15] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen, "An analytical approach to floorplan design and optimization," *IEEE Transactions on CAD*, pp. 761–769, 1991.

[16] C. Cheng and E. S. Kuh, "Module placement based on resistive network optimization," *IEEE Transactions on CAD*, pp. 218–225, 1984.

[17] H. Onodera, Y. Taniguchi, and K. Tamaru, "Branch-and-bound placement for building block layout," *Proceedings of 28th Design Automation Conference*, pp. 433–439, 1991.

[18] J. Xu, P. N. Guo, and C. K. Cheng, "Sequence-pair approach for rectilinear module placement," *IEEE Transactions on Circuits and Systems*, pp. 484–493, 1999.

[19] J. P. Blanks, "Near optimal placement using a quadratic objective function," *Proceedings of 22nd Design Automation Conference*, pp. 609–615, 1985.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[21] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," *Proceedings of 23th Design Automation Conference*, pp. 101–107, 1986.

[22] C. Sechen, "Chip-planning, and global routing of macro/custom cell integrated circuits using simulated annealing," *25th ACM/IEEE Design Automation Conference*, pp. 73–80, 1988.

[23] J. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE Journal on Solid-State Circuits*, pp. 330–342, 1991.

[24] E. Charbon, E. Malavasi, U. Choudhury, and A. Sangiovanni-Vincintelli, "A constraint-driven placement methodology for analog integrated circuits," *Proceedings of CICC*, pp. 2821–2824, 1992.

[25] T. Serdar and S. Sechen, "Akord: Transistor level and mixed transistor/gate level placement tool for digital data paths," *Proceedings of ICCAD'99*, pp. 91–97, 1999.

[26] J. P. Cohoon and W. D. Paris, "Genetic Placement," *IEEE Transactions on CAD*, pp. 956–964, Nov. 1987.

[27] S. Nakatake, H. Murata, K. Fujiyoshi, and Y Kajitani, "Bounded-slicing structure for module placement," *VLSI Design Techniques*, pp. 19–24, 1994.

[28] D. Vahia and M. Ciesielski, "Transistor Level Placement for Full Custom Datapath Cell Design," *International Symposium on Physical Design*, pp. 158–163, 1999.

[29] M. A. Riepe and K. A. Sakallah, "Transistor level micro-placement and routing for two-dimensional digital vlsi cell synthesis," *Proceedings of International Symposium on Physical Design*, pp. 74–81, 1999.

[30] A. Gupta and J. P. Hayes, "Optimal 2-d cell layout with integrated transistor folding," *Proceedings of International Conference on Computer Aided Design*, pp. 128–135, 1998.

[31] J. Kim and S Kang, "An efficient transistor folding algorithm for row-based cmos layout design," *Proceedings of 34th Design Automation Conference*, pp. 456–459, 1997.

[32] Ilog CPLEX, *Integer Linear Programming with Cplex*, Ilog, 1999.